WL-TR-91-1069

ADA COMPILER EVALUATION CAPABILITY

Tom Leavitt
Kermit Terrell

Boeing Military Airplanes
P O Box 7730
Wichita KS 67277-7730

July 1991

Interim Report for Period November 1989 - November 1990

Approved for public release; distribution is unlimited.

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6543

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) Boeing Document Number D500-12482-1	5. MONITORING ORGANIZATION REPORT NUMBER(S) WL-TR-91-1069
6a. NAME OF PERFORMING ORGANIZATION Boeing Military Airplanes 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Avionics Directorate (WL/AAAF) Wright Laboratory
6c. ADDRESS (City, State, and ZIP Code) Post Office Box 7730 Wichita KS 67277-7730	7b. ADDRESS (City, State, and ZIP Code) WPAFB OH 45433-6543
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Ada Joint Program Office 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-86-C-1059
8c. ADDRESS (City, State, and ZIP Code) Room 3E114 (1211 S. Fern Street) The Pentagon Washington DC 20301-3080	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 63756D · PROJECT NO. 2853 · TASK NO 01 · WORK UNIT ACCESSION NO. 03

11. TITLE (Include Security Classification)
The Ada Compiler Evaluation Capability
Final Technical Report Release 2.0

12. PERSONAL AUTHOR(S)
Tom Leavitt, Kermit Terrell

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM 8Nov89 TO 8Nov90	July 1991	24

16. SUPPLEMENTARY NOTATION
ACEC Version 3.0 is currently under development

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Evaluation, Performance, Usability, Test Suite, Ada, Compiler

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document reviews the findings and lessons learned in accomplishing the development of the Ada Compiler Evaluation Capability Version 2.0. This version added 300 new performance tests, assessors for program library systems, symbolic debuggers, and system diagnostics, a new tool to simplify the preparation of input to median and a Single System Analysis Tool. This report focuses on the improvements made to the testsuite, details the technical advantages of the improvements and provides a list of enhancements under consideration for ACEC Version 3.0.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. [✓ DTIC USERS]	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Raymond Szymanski	22b. TELEPHONE (Include Area Code) (513)255-3947 · 22c. OFFICE SYMBOL WL/AAAF

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

ABSTRACT

This Final Technical Report for the Ada Compiler Evaluation Capability reviews the findings and lessons learned in accomplishing Phase 3 of the project.

ACEC
Final Technical Report

Contents

1 SCOPE 5
1.1 IDENTIFICATION 5
1.2 PURPOSE 5
1.3 INTRODUCTION 5

2 APPLICABLE DOCUMENTS 7
2.1 GOVERNMENT DOCUMENTS 7
2.2 NON-GOVERNMENT DOCUMENTS 7

3 ANALYSIS OF THE TEST SUITE 8
3.1 CONFIRMED EXPECTATIONS 10
3.1.1 MEDIAN 10
3.1.2 Single System Analysis 10
3.1.3 Conventions For Valid Portable Test Problems 10
3.1.4 INCLUDE 11
3.1.5 Timing Loop Code 11
3.1.6 FORMAT-MEDIAN Interface 13
3.1.7 Guides 14
3.1.8 Assessors 14
3.1.9 Portable Math 15
3.2 PROBLEMS ENCOUNTERED 16
3.3 MEDIAN ANALYSIS LIMITATIONS 18
3.4 POSSIBLE ENHANCEMENTS 19

4 SUMMARY 22

5 NOTES 23
5.1 ABBREVIATIONS AND ACRONYMS 23

1 SCOPE

This section identifies the Ada Compiler Evaluation Capability (ACEC) Release 2.0 product, states its purpose, and summarizes the purpose and contents of this Final Technical Report.

1.1 IDENTIFICATION

This is the Final Technical Report for the ACEC Release 2.0. It was developed by Boeing's ACEC group under contract to the Wright Research and Development Center.

The first release of the ACEC program developed a Software Product consisting of a suite of benchmark test programs, support tools, a Reader's Guide, a User's Guide, and a Version Description Document (VDD). The second release of the ACEC corrected problems found in the first release; added approximately 300 new performance tests, assessors for program library systems, symbolic debuggers, and system diagnostics, a new tool to simplify the preparation of input to MEDIAN, and a Single System Analysis (SSA) tool; and upgraded supporting documentation to permit a user to assess the performance of Ada compilation systems.

1.2 PURPOSE

The purpose of this document is to review the problems encountered and the lessons learned in the process of developing the second release of the ACEC Software Product.

The descriptions on how to use the product have been presented in the Guides. Information contained in the Guides will not be repeated here.

The numeric results of the ACEC Release 2.0 testing are presented in the ACEC Software Test Report for Release 2.0 and will not be repeated here.

1.3 INTRODUCTION

The ACEC group in Boeing Wichita's Engineering Software and Languages Department tested the ACEC Release 2.0 in November - December 1989. The ACEC was tested on 4 of the same 5 trial compilation systems (but with upgrades) that were used in the first release of the ACEC; DEC, TeleSoft, 1750A cross compiler (these first 3 were VAX hosted), and the Apollo. The Silicon Graphics was used this time in place of the Harris compilation system.

The ACEC Release 2.0 Software Product consists of:

- A suite of performance test programs
- A set of supporting packages

  These include programs to test the accuracy of elementary math functions, programs to test the accuracy of the system clock, programs to include the timing loop in the test programs, and a portable implementation of a math library.

- A set of analysis tools

  These include FORMAT, MED_DATA_CONSTRUCTOR, MEDIAN and Single System Analysis (SSA)

- A set of assessors for diagnostics, symbolic debuggers, and program library systems

- Sample command files to compile and execute the tools and tests for a VMS and for a UNIX based system which can be used as guides for porting to other systems

This report assumes that the reader is familiar with the ACEC Release 2.0 User's Guide, the Reader's Guide, and the Version Description Document (VDD). Individual test problems are referred to by name (in capitals) in this report. The reader should refer to the VDD Release 2.0 for a description of the problem. Information from the Software Test Report Release 2.0 is also referred to.

This document reviews the development of the second release of the test suite and supporting tools. It gives an analysis of results and a review of the lessons learned.

2 APPLICABLE DOCUMENTS

The following documents are referenced in this guide.

2.1 GOVERNMENT DOCUMENTS

MIL-STD-1815A	Reference Manual for the Ada Programming Language (LRM)

2.2 NON-GOVERNMENT DOCUMENTS

D500-12470-1	Ada Compiler Evaluation Capability (ACEC) Technical Operating Report (TOR) User's Guide Release 2.0 Boeing Military Airplanes P.O. Box 7730 Wichita, Kansas
D500-12471-1	Ada Compiler Evaluation Capability (ACEC) Technical Operating Report (TOR) Reader's Guide Release 2.0 Boeing Military Airplanes
D500-12472-1	Ada Compiler Evaluation Capability (ACEC) Version Description Document (VDD) Release 2.0 Boeing Military Airplanes
D500-12467-1	Ada Compiler Evaluation Capability (ACEC) Software Test Report Release 2.0 Boeing Military Airplanes
	Understanding Robust and Exploration Data Analysis, by Hoaglin, Mosteller, and Tukey John Wiley & Sons, Inc, 1983

3 ANALYSIS OF THE TEST SUITE

The numeric values obtained by executing the second release of the test suite were presented in the Software Test Report Release 2.0 and will not be duplicated here.

The following table displays the test problems where one of the quartiles was flagged by MEDIAN as exceptional. Such highly variable test problems might suggest cases where systems are not performing comparable operations. These lines were extracted verbatim from the MEDIAN report.

```
TEST PROBLEM
NAME          |  MIN    LOWER   MEDIAN  UPPER    MAX   | SPREAD
----------------------------------------------------------------
IO16          | 0.27-   0.34-   1.00    2.90+   3.81+  |  8.41
IO20          | 0.27-   0.23-   1.00    3.58+   3.67+  | 12.79
IO24          | 0.15<   0.25-   1.00    3.96+  22.48>  | 15.71
IO25          | 0.04<   0.25-   1.00    3.93+   4.59+  | 15.45
IO26          | 0.29-   0.30-   1.00    3.33+  26.45>  | 11.10
IO27          | 0.21-   0.29-   1.00    3.51+  25.28>  | 12.30
IO28          | 0.07<   0.37-   1.00    2.68+   6.59>  |  7.17
IO30          | 0.07<   0.38-   1.00    2.66+   6.57>  |  7.10
IO_80_20_1    | 0.20-   0.28-   1.00    3.63+   4.01+  | 13.17
IO_80_20_2    | 0.12<   0.17<   1.00    5.90>   6.37>  | 34.80
IO_80_20_3    | 0.09<   0.33-   1.00    3.04+   3.57+  |  9.22
IO_INTER3     | 0.10<   0.13<   1.00    7.73>  11.20>  | 59.82
IO_PATTERN2   | 0.07<   0.15<   1.00    6.67>   9.51>  | 44.49
IO_PATTERN3   | 0.07<   0.16<   1.00    6.13>   8.80>  | 37.52
IO_PATTERN4   | 0.08<   0.17<   1.00    5.73>   9.48>  | 32.87
IO_PATTERN6   | 0.05<   0.12<   1.00    8.21>  13.52>  | 67.45
IO_PATTERN7   | 0.05<   0.13<   1.00    7.74>  12.04>  | 59.93
IO_PATTERN8   | 0.06<   0.13<   1.00    7.42>  12.81>  | 54.99
IO_RECUR1     | 0.12<   0.21-   1.00    4.72+   6.68>  | 22.27
IO_RECUR2     | 0.21-   0.28-   1.00    3.62+   9.24>  | 13.13
IO_RECUR3     | 0.27-   0.35-   1.00    2.84+   7.19>  |  8.06
IO_SCAN4      | 0.07<   0.16<   1.00    6.31>  10.35>  | 39.77
IO_SCAN8      | 0.05<   0.12<   1.00    8.49>  14.77>  | 72.01
SS153         | 0.06<   0.35-   1.00    1.21    4.95+  |  3.41
SS602         | 0.06<   0.37-   1.00    1.21    5.12+  |  3.22
SS753         | 0.09<   0.35-   1.00    1.28    5.61>  |  3.62
SS754         | 0.09<   0.37-   1.00    1.25    5.57>  |  3.35
SS802         | 0.03<   0.24-   1.00    1.19    1.43   |  5.04
```

The following list describes the features for each flagged test problem responsible for the performance variations as identified by the ACEC group.

TEST PROBLEM NAME	FEATURES
IO16	References SIZE function of a DIRECT file. Performance depends on underlying operating system. Some may not require disc reference.
IO20	References END_OF_FILE function of a SEQUENTIAL file. Performance depends on underlying operating system. Some may not require disc reference.
IO24	Console I/O device dependent
IO25	Console I/O
IO26	Console I/O
IO27	Console I/O
IO28	Console I/O
IO30	Console I/O
IO_80_20_1	Direct file random reads from a small set of records
IO_80_20_2	Direct file random reads from a small set of records
IO_80_20_3	Direct file random reads from a small set of records
IO_INTER3	Direct file random read/write in a small set of records
IO_PATTERN2	Direct file random reads from a small set of records
IO_PATTERN3	Direct file random reads from a small set of records
IO_PATTERN4	Direct file random reads from a small set of records
IO_PATTERN6	Direct file random writes to a small set of records
IO_PATTERN7	Direct file random writes to a small set of records
IO_PATTERN8	Direct file random writes to a small set of records
IO_RECUR1	Direct file random reads from a small set of records
IO_RECUR2	Direct file random reads from a small set of records
IO_RECUR3	Direct file random reads from a small set of records
IO_SCAN4	Direct file random reads from a small set of records
IO_SCAN8	Direct file random writes to a small set of records
SS153	Raises exceptions
SS602	Raises exceptions
SS753	Raises exceptions
SS754	Raises exceptions
SS802	Call on CALENDAR.SECONDS

3.1 CONFIRMED EXPECTATIONS

The following subsections list design decisions which worked well.

3.1.1 MEDIAN

If comparing multiple systems, some type of statistical analysis, similar to MEDIAN, is necessary for practical use of a test suite with more than a handful of test problems. The analysis focuses the attention of the ACEC users on the test problems with "unusual" results. It permits a form of report-by-exception where ACEC users can concentrate their efforts on exploring the test problems with anomalous performance. Since most test problems will not be flagged by MEDIAN as outliers, ACEC users will be able to "skim" over most of them and concentrate on those where large differences between systems were observed.

Without an analysis tool, a test suite would require users to "understand" each of the test problems, at least to know if a result on one system was good, bad, or indifferent. The residual matrix is very helpful since it establishes a normalized metric for test results — a residual close to one is "typical." No analysis tool can extract more information from a set of data than is implicit in the collection of "raw" measurements, but it can make the relationships between data more apparent. It would be very easy to overlook the fact that a system executes some test problems twice as fast as typical when all the data is presented as one large table of timing measurements. It is important to prevent users from being overwhelmed by the volume of data and MEDIAN serves this role.

3.1.2 Single System Analysis

The Single System Analysis tool provides an automated way of comparing and displaying results from sets of related performance test problems. The ACEC team has manually compared results of related tests before the development of this tool, however, the speed and automated nature of the tool now make it much simpler to do. This ease of use encourages ACEC users to obtain the full benefits of collecting the performance data.

The Single System Analysis tool is straightforward in principle and no unanticipated problems arose during its development and use other than on the Silicon Graphics. The system appears to have a problem with exception handling.

3.1.3 Conventions For Valid Portable Test Problems

Writing valid portable benchmarks is not easy. It is necessary to be careful in constructing test problems so that they are not unduly optimizable. This was anticipated and confirmed past experience. The conventions established for developing tests were generally effective in precluding unexpected optimizations.

3.1.4 INCLUDE

INCLUDE is an ACEC support tool used to textually expand Ada source text It is used to insert the timing loop code into the test programs. Refer to the User's Guide for a discussion on the use of this tool

The decision to use a separate tool to INCLUDE the timing loop code has worked well. It has permitted flexible modification of the timing code

- To modify the basic timing loop code as needed
- To switch between Central Processing Unit (CPU) time and elapsed time measurements
- To accommodate implementation dependencies (such as the GETADR calls to measure code expansion size when the label'ADDRESS attribute did not work)

3.1.5 Timing Loop Code

The timing loop code is responsible for measuring the execution time and code expansion for each test problem It is discussed in depth in the Reader's Guide, including the constraints on its design and the conventions on writing test problems so that the test designer can measure what was intended

The timing loop code was enhanced from the first release in several ways

1 The termination condition for the inner timing loop has incorporated a statistically robust confidence test adapted from Understanding Robust and Exploration Data Analysis, by Hoaglin, Mosteller, and Tukey An unreliable measurement indicator is printed when either a t-test or an r-test fails — the outer timing loop uses the t-test to terminate with consistent measurements because this is a more conservative test

   This has been a valuable change because it reduces the number of test problems which are unnecessarily reported as unreliable measurements.

2 Most of the code associated with the timing loop in STOPTIME0 has been moved into a procedure in GLOBAL

   This is valuable because it reduces the size of the generated code for multiple problem test programs and reduces compilation time It saves ninety-five lines of code per problem

3 The printing of time measurements has been modified so that an exponential format is used for values less than ten microseconds or greater than one hundred seconds.

   This is valuable because using the scheme from the first release, some of the small test problems on fast machines would be truncated to one significant digit when printed, making the effort expended to calculate measurements to a five per cent confidence interval unnecessarily precise The processing for large times is also necessary (some of the I/O test problems developed for the second release product execute for more than 100 seconds on some of the trial systems) so that FORMAT can recognize the large times.

4 Sometimes measurement noise makes the timing loop code calculate a small positive time for test problems which have been optimized into a null. The timing loop was modified to check whether the code expansion size measurements indicate that the test problem has been optimized into a null and to print a zero time in this case

5 In the first release, the ACEC timing loop reported all test problems with "uncorrected" execution times less than the null loop time as zero. The timing loop was modified to test for the case where subtracting the null loop time from the total problem execution time to calculate the effective test problem execution time would produce a negative value which is larger than can be attributed to measurement noise. If the computed negative time is "small" (less than the variations in the null loop during initialization of the timing loop in executing each test program), it is reported as zero. A "large" negative value will result in an unreliable measurement error code.

This change was valuable because it tells users when unusual behavior is observed.

6 The timing loop was modified to detect when a null statement is being evaluated and to not perform the maximum number of iterations in this case.

The benefit of this change is that it saves some execution time when running test problems optimized into null statements.

7 The timing loop (INITTIME) was modified to explicitly test whether the system has taken special steps to return a unique value for CALENDAR.TIME each time it is called. Some operating systems have implemented the time function such that every value is guaranteed to have a unique value during each day — where their actual clock resolution is not as accurate as would be necessary to satisfy this system requirement with the clock value, they simply track the number of calls on the TIME function since the actual clock "ticked" and return as the function result the actual time plus the number of calls (having ensured that on their fastest system, the number of calls which can be made on the TIME function before the clock "ticks" is less than the resolution of the return type) On these systems, the low order bits are essentially useless as indicators of actual time. On first encountering this behavior, many programmers may think it is a strange way for a system to keep time, however it does have utility — some low-level programs are simplified if they can assume that time-stamps are unique and form a monotonic sequence. It is typical for Ada systems to "pass-through" the operating system function for the TIME function, so that this behavior is not hidden from Ada programs. The first released ACEC timing loop could calculate inaccurate times — such systems render the software vernier calibration process that the ACEC uses futile. The second release

of the ACEC tests on initialization for this behavior and if detected, uses a large value for MIN JITTER COMPENSATION in the timing loop to essentially make the ACEC calculations not rely on the vernier.

This change is valuable because if it were not compensated for, the ACEC timing loop could produce inaccurate estimates and error bounds on some systems for fast executing test problems.

3.1.6 FORMAT-MEDIAN Interface

The execution of the ACEC test suite produces a log file containing performance measurements for each test run. The log file must be processed before the measurement data can be input to MEDIAN. The method used in the first release to interface the results file and the MEDIAN analysis tool was awkward. The first release required the user to execute FORMAT to extract a data aggregate, use a text editor to insert the aggregate into MED_DATA, and then recompile MED_DATA and MEDIAN.

For the second release, the MED DATA CONSTRUCTOR tool was written to automate the process of creating a MED_DATA package from the outputs of FORMAT from different systems. The user must execute FORMAT to create data aggregates and then execute MED DATA CONSTRUCTOR to create a MED DATA package containing data for the specified systems. MED DATA and MEDIAN are then compiled, and MEDIAN is executed. The user must use a text editor to create a file containing the system names and the names of the data files to be used by MED_DATA_CONSTRUCTOR. The user is not required to edit the data aggregates.

Another approach would have been to combine in one program the functions of FORMAT, MED_DATA_CONSTRUCTOR, and MEDIAN. This was not done because the ACEC interface must deal with bare machines which have limited I/O capabilities.

The method selected permits the collection and processing of results on target machines which have only a console output device (that is, no way to write to a file for processing by another program). This constraint limits the design options and the degree of "friendliness" possible — the ACEC would be friendlier if each test program updated a file to reflect results, then MEDIAN (or other analysis tools) could read this file.

The current interface provides some important advantages.

1. The order in which a user executes test programs is not important.

2. Users do not have to create or update a large positional array. It would be error prone to manipulate an aggregate with over a thousand entries — the risks of recording results from one test problem as coming from another test problem would be large. Aggregates with named associations are helpful here.

3. It is a flexible system. It is simple for a user to incorporate additional test programs into the suite, as long as they avoid naming conflicts with existing test problems. It is simple for a user to use MEDIAN to analyze subsets of the data, such as only the tasking tests.

3.1.7 Guides

The writing of effective user documentation is difficult and time consuming. It is also very important to the usability of a product, particularly when there is no provision for telephone "hot-lines" for helping users who run into problems.

The ACEC user documentation is extensive (over 400 pages) and comprehensive.

The User's Guide and Reader's Guide present an introduction to the issues involved in measuring performance. They discuss:

- Compiler optimization
- Aspects of machine design which influence performance
- Pitfalls in measuring system performance
- Statistics of collecting and analyzing performance data

They provide helpful background information to ACEC users about performance evaluation in general and Ada in particular. They also provide detailed instructions for using the ACEC.

The ACEC Version Description Document (VDD) contains useful information about the test suite itself and the individual test problems.

The general reaction to the first release of the ACEC documentation was favorable.

3.1.8 Assessors

Release 2 of the ACEC includes new assessors for the Ada program library management system, the diagnostic messages, and the symbolic debugger. The assessors, especially the debugger assessor, were found to be more time-consuming to execute than originally predicted. This is due to two factors:

- Adaptation effort.

  The assessors test a wide range of capabilities and a large number of individual actions which must be separately adapted to each system. The effort required is significant, even when the evaluator is familiar with the product.

  The program library assessor includes 19 scenarios with 80 separate questions.

  The diagnostic message assessor includes 45 scenarios with 455 separate questions.

  The symbolic debugger assessor includes 29 scenarios with 118 separate questions.

- Learning time.

  There was a learning time required for an evaluator to become familiar with a system not previously used.

  In order to perform the library assessor and the debugger assessor, the evaluator must learn implementation specific program library and debugger commands and concepts. It is necessary to adapt the commands used to execute each scenario. The assessors cover a broad scope, and before deciding that some capability was not supported, it was necessary to stop and review manuals to make sure that something had not been overlooked in the initial learning process.

  The initial projections for learning time were based on the assumption that the trial systems would be similar to the system the assessors were developed on. This was overly optimistic.

  We believe that ACEC users will find the effort worthwhile because it will provide them with more organized information about the capabilities of systems than they would acquire with a similar amount of time spent in unstructured experimentation. Although vendors can tell their users that their library system and/or the debugger provide all the useful capabilities, running the ACEC assessors will assure the testers that the capabilities exist and that they know how to use them.

3.1.9 Portable Math

The second release of the ACEC added a representation independent version of MATH_DEPENDENT, greatly increasing the portability of the ACEC provided generic math library and therefore, simplifying the programmer effort required to produce a working math library.

There was one trial system which provided a library which was not usable for an interesting reason. That system named the provided math library "MATH" — the same name ACEC uses for its math package. This presents a naming conflict which could not be resolved without extensive modifications to the source text of the ACEC test programs and analysis tools or modifications to the source of the implementation provided math library. Since neither of these alternatives was acceptable, on this system testing proceeded using the ACEC provided portable math library.

The ACEC math package specification (and all the programs referencing the specification) were changed to be compatible with a subset of the Association for Computing Machinery. Special Interest Group on Ada, Numerics Working Group (NUMWG) recommendations. This change may be of increasing value in the future if and when more implementations adapt the NUMWG recommendations. Only one of the trial systems provided a NUMWG implementation, and on this trial system there were no problems with using the vendor provided implementation in place of the ACEC provided math package.

3.2 PROBLEMS ENCOUNTERED

During the development of the ACEC, the test suite and tools were executed on multiple systems both to verify portability of the code, and to provide sample data to demonstrate the comparative analysis tools. These are the "trial" systems referred to throughout this report. During this process some implementation errors and restrictions were discovered in the trial systems, as detailed below:

- Some systems did not support the label'ADDRESS clause (used in the ACEC to measure code expansion size).
- Most systems do not support tying tasks to interrupts (a Chapter 13 feature).
- Most systems do not support a type'SMALL specifying a fixed point delta which is **not** a power of two.
- Most systems did not support asynchronous I/O operations — that is, an I/O operation in a task causes the *program* to halt until the I/O completes.
- A few systems failed to always reclaim implicitly allocated space. This prevented some test problems from completing. The LRM does **not** require that the space be reused, so this is an implementation restriction rather than an error.
- Some systems restrict the type of files that can be processed. Test problems using a SEQUENTIAL_IO instantiated with an unconstrained type (that is, STRING) were not accepted by many systems, as were problems using sequential and direct files with a variant record. This feature is not supported by many of the trial systems. On the DEC Ada system the test problems would not run using the default FORM strings and a system dependent adaptation was necessary.
- The test problem (SS747) which calls on an assembly language procedure did not work on all systems. In several cases, the trial systems claim to support the facility, but the provided documentation was unclear, and the initial attempt to adapt the test problem failed.
- Capacity limitations — some programs were too large to be compiled by some of the systems.
- The implementation dependent type SYSTEM.ADDRESS is converted to an integer type to calculate sizes. The size of this integer type is implementation dependent and may need to be adapted for use on different systems.
- Some implementations imposed restrictions on length clauses. Several only supported the declaration of specific predefined sizes. For example, several would not accept a specification for a three bit wide field.

- Preemptive priority scheduling was not supported on all the trial systems. This caused some of the test problems to report a runtime error.

- Some compilation systems did not support integer types which required more than 16 bits. This restriction prevented some test programs from operating.

- One system (for an embedded target) did not support any file I/O.

- Some systems did not support an option to specify tasking discipline (time-sliced vs run-till-blocked). This is not required but tests for it were included in the second release.

- Particularly with the Silicon Graphics system, there were test problems which were measured as taking zero time but were not NULL statements (they had positive code expansion sizes, performed non-optimizable operations, and code was generated). In some cases, when the test programs were rerun, the spurious zero measurements did not reoccur, but this was not universal.

  Initial investigation is inconclusive and more testing with the system would be required to isolate the source of the problem, which might be applicable to other compilation systems.

Although the portability of the test suite could have been improved by restricting it to use only the features supported by the most limited of the known Ada implementations, this was not done. It is important to evaluate all the capabilities of the language. Some of the features not supported by all systems are important to some users and need to be covered in a comprehensive test suite.

3.3 MEDIAN ANALYSIS LIMITATIONS

One potential criticism of the ACEC product is that the system factors computed by MEDIAN will not reflect the performance of Ada implementations on the types of applications all users are interested in. MEDIAN computes system factors by averaging over the test problems. If the distribution of language features is very different in a user's application than the distribution in the ACEC test suite, the system factors may not accurately predict performance of different systems on the user's ultimate application. For example, comparing two systems, one of which provides floating point support by software simulation and one with hardware support, the system with software floating point will be slower. For a user whose applications will not use floating point, the differences due to floating point speed are not important and such a user would prefer that the speed of floating point operations not influence the evaluation of systems.

The above situation reflects the fact that the system factors are biased by the selection of test problems. There are several issues involved here:

1. Some limitation is unavoidable. If there was only one test problem, critics could complain that the problem does not represent the usage of language features they anticipate for their applications. If there are multiple test problems, covering different language features, critics could complain that the combination of the test problems does not reflect their anticipated usage.

2. The limitation is not unreasonable. Users with specific interests may find information about them — 16 or 32 bit integer operations, execution with or without constraint checking suppressed, etc.

   The ACEC makes heavy use of variables declared in library packages (as contrasted to variables declared in local scopes). This reflects our expectations of typical usage. The ACEC also contains test problems using locally declared variables so differences between scopes can be observed. The fraction of operations on integer, floating-point, and fixed-point types, and the precisions of the operations are application sensitive. The computed system factors will reflect the distribution of usages in the test suite. Users interested in the performance of specific language features can examine the test problems using these features, by reference to the VDD appendices. If some language features are particularly slow on one system, MEDIAN will flag test problems using this feature as outliers, drawing attention to the performance issue.

3. It is easy for ACEC users to select a subset of test problems (perhaps including additional test problems they develop themselves) to reflect their expected usage. For example, a user may decide to ignore all the file I/O test problems if their project is targeted for a bare machine; for applications targeted to a general purpose multiuser system, it may be best not to consider the interrupt test problems.

3.4 POSSIBLE ENHANCEMENTS

Several possible enhancements to the ACEC product have been mentioned at conferences by ACEC users. Some of these have been included in the second ACEC release. Those that are not yet available are defined in the following paragraphs.

- Capacity testing

  The ACEC does not include any explicit tests designed to systematically determine capacity limitation at compile, link, or execution time. Some implicit capacity testing is inherent in the organization of the test suite and the assumption that a system will be able to compile and execute the programs in the ACEC Software Product. However, it would be useful to incorporate explicit capacity testing.

- Systematic compile speed testing

  The ACEC provides for the recording and analysis (through MEDIAN) of the elapsed time to compile and link the performance test programs. However, these test programs are not designed to contain selections of language features (distributions of features) expected to cause variations in compile/link times. It would be useful to incorporate test programs explicitly designed to emphasize features expected to result in different compile times.

- Memory size

  It has been suggested that the ACEC be enhanced to include programs to test execution time memory capacity limits.

- Additional problems

  Although large, the ACEC is not exhaustive in its testing. Problems of interest to specific application areas could be added to increase the utility of the ACEC.

- Reliability of timing measurements

  The ACEC timing loop code has evolved during the development of the product, and there are several enhancements which could be explored in order to improve its ability to efficiently produce reliable, portable timing measurements. Of particular concern is the tendency on one of the trial systems for unreliable codes to be reported rather than time measurements — it may be valuable to experiment with increasing the maximum number of outer timing loop cycles (GLOBAL.MAX_ITERATION_COUNT and associated variables, including the values of the array GLOBAL.T_VALUE).

- User interface

  Several users have criticized aspects of the ACEC user interface, as listed below:

- Multiple problem programs

  Some of the ACEC performance test programs contain multiple problems. When one problem in a program fails to compile, no results are obtained for any of the problems in that program unless the ACEC user modified the source program to correct (or remove) the failing problem. It has been suggested that each test problem be presented as a separate compilation unit.

- Awkward data interface

  The FORMAT - MEDIAN interface in release one has been criticized as awkward and the time to re-analyze data as long. If MEDIAN were to read a set of FORMAT aggregates, analysis of subsets of data would be quicker and simpler to perform. The MED_DATA_CONSTRUCTOR tool introduced with the second release should help.

- MATH package

  The first release of the ACEC was criticized because significant user effort was required to adapt the MATH package to a new target. Three steps taken in the second release should greatly ease this effort.

  * In the second release, package MATH is NUMWG compatible, making it easy to use an implementation provided math library on targets which support the NUMWG recommendations.
  * A representation independent implementation of package MATH DEPENDENT is provided which makes porting the ACEC provided package GEN_MATH straightforward.
  * The User's Guide discussion of math porting issues has been expanded making it clear that the users should try to interface to an implementation provided math library where available.

• Weighting of test problems for analysis

There is no current provision for weighting test problems based on importance. Some users have requested an easy way of isolating test problems of interest to them and of analyzing performance data emphasizing specific areas. The ACEC users have had, and continue to have, the option of using MEDIAN on any subset of problems they wanted to consider. This suggestion is related to the alternative analysis techniques suggested for MEDIAN statistical processing.

• MEDIAN statistical processing

Some users have suggested that the ACEC statistical analysis technique be modified to provide for inference – confidence intervals and significant differences. There is some statistical research which is suggestive of ways to add confidence estimates while retaining robust statistical analysis, but the issue would require more study.

- User Documentation

  Some users have suggested that the ACEC documentation be expanded to include all issues which might affect compiler selection.

4 SUMMARY

Significant effort was expended to address criticisms of the original ACEC release. All of the reported problems with Release 1.0 were either addressed in this second release or identified as possible future enhancements. There were no major surprises with the ACEC Release 2.0 product. The ACEC found there were general improvements in the Ada compilers from Release 1.0 to 2.0. However, there was still one trial system that was barely usable due to system crashes and destroyed directories.

The new features of this release were: 300 new performance tests (I/O, implicit storage reclamation, application profile tests, etc.); assessors (library management, symbolic debugger, and diagnostic messages); the MED_DATA_CONSTRUCTOR; and the Single System Analysis (SSA) tool.

Even though approximately 300 performance tests were added, the total lines of "included" source remained constant due to a change in the timing loop code. Also, changes to the example COM files for the performance tests simplified porting. For the new I/O performance tests, both the file and console I/O tests showed large differences between systems – which was anticipated. For the new implicit storage reclamation performance tests, it was surprising to find failing systems.

The assessors evaluate features that were believed to be the most valuable for day to day usage, but not all of these capabilities are essential to have a usable system. The Diagnostics Assessor proved to be straightforward to execute and easy to adapt. User feedback on its grading system will give an idea as to its usefulness. The Library and the Symbolic Debugger Assessors required more adaptation effort than was anticipated. Some systems do not support many of the extensive capabilities explored in these assessors, but the assessors were designed with future technology in mind. One system, the 1750A cross compiler, did not have a debugger. Most of the trial systems' program libraries were more fragile than anticipated.

The MED DATA CONSTRUCTOR program was an improvement to the FORMAT - MEDIAN interface. This program worked as anticipated by significantly reducing the manual effort needed to format the data for the MEDIAN analysis.

The SSA did not provide any new information, but organized the available data and presented it so that the user did not have to tediously compare related tests. By providing a tool which automates the comparisons of related tests, the SSA will give more insight into system performance with less user effort than when the comparisons had to be made by hand.

To execute this ACEC Release 2.0 test suite and assessors through the analysis phase is not a trivial task. A lot of time, effort and computer resources will be necessary in order to extensively test an Ada compiler. But, when a user has completely run the ACEC test suite, assessors and analysis tools, the majority of a compilation system's features will have been exercised and the evaluator will have a good, overall knowledge of that compiler and associated capabilities.

5 NOTES

This section contains information only and is not contractually binding.

5.1 ABBREVIATIONS AND ACRONYMS

ACEC	Ada Compiler Evaluation Capability
ACVC	Ada Compiler Validation Capability
BMA	Boeing Military Airplanes
CPU	Central Processing Unit
I/O	Input / Output
LRM	(Ada) Language Reference Manual (MIL-STD-1815A)
NUMWG	Numerics Working Group (NUMWG) subcommittee of the Association for Computing Machinery, Special Interest Group on Ada
SSA	Single System Analysis
TOR	Technical Operating Report
VDD	Version Description Document